More Arduino Programming

- Digital I/O (Read/Write)
- Analog I/O
- Tones
- Delays

Digital I/O (Digital Pins)

- Digital pins on the Arduino can be defined as being Inputs or Outputs using the function pinMode().
- The state of a digital pin can be determined with the function digitalWrite().
- The state of an output can be set as High or Low with the function digitalWrite().
digitalWrite()...

- `digitalWrite()` writes a HIGH or a LOW value to a digital pin.
- If the pin has been configured as an OUTPUT with `pinMode()`, its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.
- If the pin is configured as an INPUT, `digitalWrite()` will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the `pinMode()` to INPUT_PULLUP to enable the internal pull-up resistor. See the digital pins tutorial for more information.
  - e.g. `digitalWrite(ledPin, HIGH)`

digitalRead()...

- `digitalRead()` reads the value from a specified digital pin, either HIGH or LOW.
- e.g. `digitalRead(inPin);`

Example: Blinking an LED

```c
const int LED = 10;
int blinks = 5; // blink 5 times;
bool done = false;
void setup()
{
  pinMode(LED, OUTPUT); // set pin 10 as an OUTPUT
  digitalWrite(LED, LOW); // Initialize off
}
void loop()
{
  while (done != true)
  {
    for (int i = 1; i <= blinks; ++i) // ++i same as i = i+1
    {
      digitalWrite(LED, HIGH); // Turn on LED
      delay(500); // Pause
      digitalWrite(LED, LOW); // Turn off LED
      delay(500); // Pause
    }
    done = true;
  }
}
```
Example: Debouncing a Pushbutton

```cpp
const int LED = 9, BUTTON = 2;

bool lastState = LOW, currentState = LOW, lit = false;
void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(BUTTON, INPUT);
}

boolean debounce(boolean last) //function
{
    boolean state = digitalRead(BUTTON);
    if(last != state) // has button settled down
    {
        delay(5); //delay if not
        state = digitalRead(BUTTON); //and read again
    }
    return state;
}

void loop()
{
    currentState = debounce(lastState); //call function
    if (lastState == LOW && currentState == HIGH)
    {
        lit = !lit;     //toggle LED
    }
    lastState = currentState;
    digitalWrite(LED, lit); //update LED
}
```
**analogRead()…**

- `analogRead()` reads the value from the specified analog pin.
- The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using `analogReference()`.
- It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

**analogWrite()…**

- `analogWrite()` writes an analog value as pulse width modulation (PWM) to a pin.
- This is a technique for getting analog results with digital means.
- Digital control is used to create a square wave (0 to 5 VDC).
- You can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends “on” versus the time that the signal spends “off”.
- The duration of “on time” is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5 VDC controlling the brightness of the LED.
PWM

- Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalWrite() on the same pin).
- The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

analogReference()

- analogReference() - configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options are:
  - DEFAULT: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
  - INTERNAL: an built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (not available on the Arduino Mega)
  - INTERNAL1V1: a built-in 1.1V reference (Arduino Mega only)
  - INTERNAL2V56: a built-in 2.56V reference (Arduino Mega only)
  - EXTERNAL: the voltage applied to the AREF pin (0 to 5V only) is used as the reference.
analogReadResolution()…

- analogReadResolution() is also an extension of the Analog API for the Arduino Due and Zero.
- Sets the size (in bits) of the value returned by analogRead(). It defaults to 10 bits (returns values between 0-1023) for backward compatibility with AVR based boards.
- The Due and the Zero have 12-bit ADC capabilities that can be accessed by changing the resolution to 12. This will return values from analogRead() between 0 and 4095.

Reading a Potentiometer

Igoe, Tom. If statement tutorial. Arduino 2012

```cpp
const int analogPin = A0;
const int ledPin = 13;
const int threshold = 400;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int analogValue = analogRead(analogPin);
  if (analogValue > threshold) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
  Serial.println(analogValue);
  delay(1);
}
```

Igoe, Tom. If statement tutorial. Arduino 2012

Reading a Potentiometer…

Igoe, Tom. If statement tutorial. Arduino 2012
**Analog Input Pins as Digital Pins**

- The analog pins can be used identically to the digital pins, using the aliases A0 (for analog input 0), A1, etc. For example, the code would look like this to set analog pin 0 to an output, and to set it HIGH:

  ```
  pinMode(A0, OUTPUT);
  digitalWrite(A0, HIGH);
  ```

- The analog pins also have pull-up resistors, which work identically to pull-up resistors on the digital pins. They are enabled by issuing a command such as:

  ```
  pinMode(A0, INPUT_PULLUP);  // set pull-up on analog pin 0
  ```

**Advanced I/O: tone()**

- `tone(pin, frequency)` - Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to `noTone()`. The pin can be connected to a piezo buzzer or other speaker to play tones.

- Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to `tone()` will have no effect. If the tone is playing on the same pin, the call will set its frequency.

- Use of the `tone()` function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

**Tone – Min/Max Frequencies**

<table>
<thead>
<tr>
<th>Board</th>
<th>Min frequency (Hz)</th>
<th>Max frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uno, Mega, Leonardo and other AVR boards</td>
<td>31</td>
<td>65535</td>
</tr>
<tr>
<td>Gemma</td>
<td>Not implemented</td>
<td>Not implemented</td>
</tr>
<tr>
<td>Zero</td>
<td>41</td>
<td>275000</td>
</tr>
<tr>
<td>Due</td>
<td>Not implemented</td>
<td>Not implemented</td>
</tr>
</tbody>
</table>
**Time – millis() and micros()…**

- **millis()** - returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.
- **micros()** - returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes.
  - On 16 MHz Arduino boards, this function has a resolution of four microseconds (i.e. the value returned is always a multiple of four).
  - On 8 MHz Arduino boards (e.g. the LilyPad), this function has a resolution of eight microseconds.

**delay() and delayMicroseconds()…**

- **delay(ms)** - Pauses the program for the amount of time (in milliseconds) specified as parameter. There are a thousand milliseconds in a second.
- **delayMicroseconds()** – Pauses in microseconds.
- Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use `delay()` instead.

**Other Arduino Boards**

Other Arduino boards available. e.g. LilyPad Arduino
Summary

- Functions:
  - Digital I/O (Read/Write)
    - Digital pins are defined as INPUT or OUTPUT and having levels of HIGH and LOW.
  - Analog I/O
  - Tones
  - Delays

- Examples reviewed:
  - Blinking an LED.
  - Debouncing a pushbutton.
  - Reading a potentiometer.